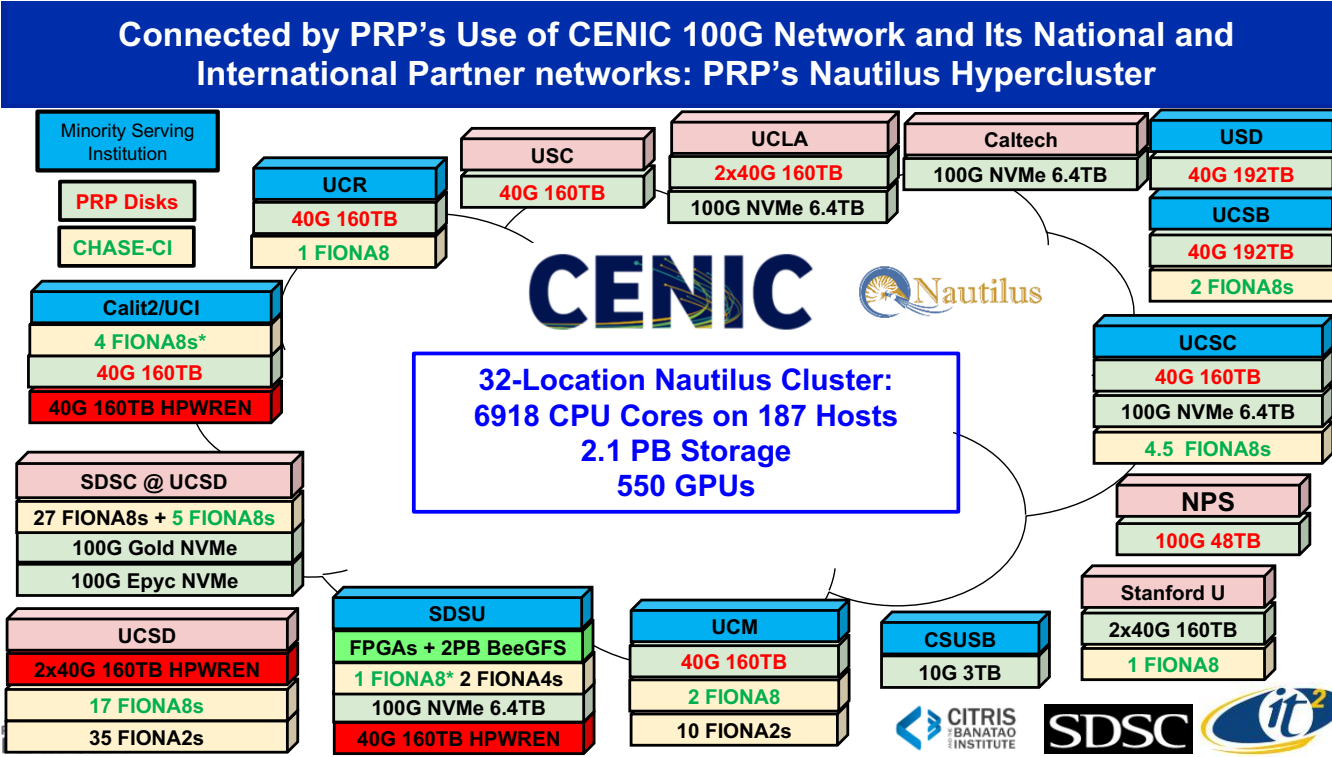


Scheduling a Kubernetes Federation with Admiralty

Igor Sfiligoi - UC San Diego
on behalf of the PRP/Nautilus group

PRP, Nautilus and Kubernetes

- The Pacific Research Platform (PRP) has been using Kubernetes since 2016
 - Started as a way to conveniently schedule network test services
 - Evolved in being a convenient platform for ML research
- OSG has had a CE gathering opportunistic cycles for over a year now
 - As well as orchestrating some of its services e.g. StashCache and Frontends



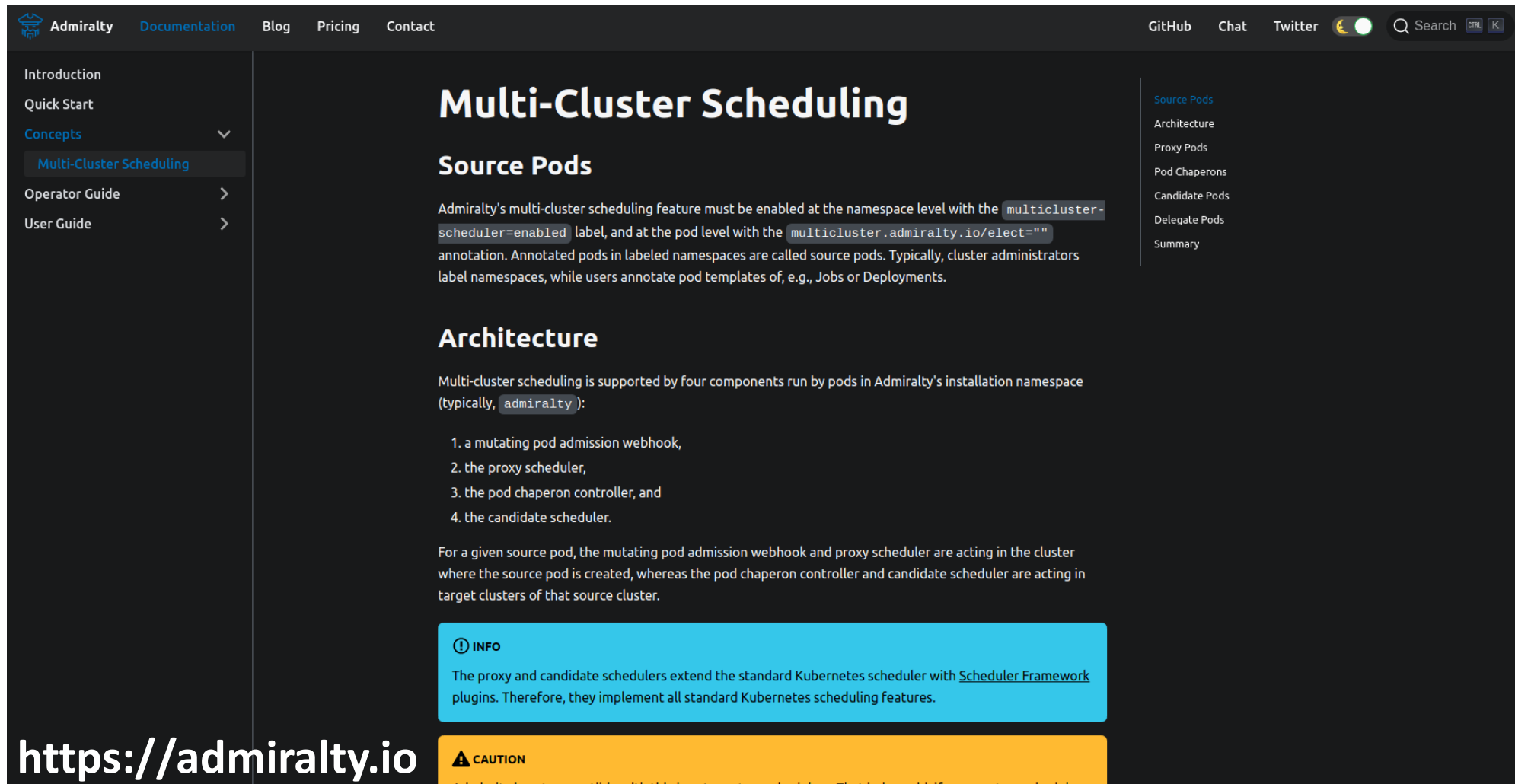
Why federation?

- PRP/Nautilus has been steadily growing
 - It now has nodes also in Asia, Europe and Australia
 - While successful, we do understand not everyone will want to join the club
- Separate administration domains
 - We even have the use case at UCSD
 - PRP Nautilus and SDSC Expanse will operate separately, but will work together through federation
- Multiple platforms
 - PRP has an IoT component, where ARM CPUs rule
 - Having a dedicated ARM k3s and federating with it ended being simpler

Driving principles

- We wanted a “native Kubernetes” solution
 - I.e. kubectl should be all that the user needs
- We did **not** want a centralized solution
 - All participating Kubernetes clusters should be on equal playing field
 - Each Kubernetes cluster should be able to participate in any number of federations
- We did not want to do any development ourselves
 - Helping with testing OK
 - Occasional patch OK
 - But no long-term maintenance

Admiralty's Multicluster-Scheduler



The screenshot displays the Admiralty documentation website. The top navigation bar includes links for Admiralty, Documentation, Blog, Pricing, and Contact. On the right, there are links for GitHub, Chat, Twitter, a dark mode toggle, and a search bar. The left sidebar contains a table of contents with sections like Introduction, Quick Start, Concepts (highlighted), Multi-Cluster Scheduling (selected), Operator Guide, and User Guide. The main content area is titled 'Multi-Cluster Scheduling' and features a 'Source Pods' section explaining the multi-cluster scheduling feature, its enabling via labels and annotations, and a list of four components: mutating pod admission webhook, proxy scheduler, pod chaperon controller, and candidate scheduler. An 'Architecture' section follows, describing the roles of these components. At the bottom, there are two callout boxes: an 'INFO' box stating that the proxy and candidate schedulers extend the standard Kubernetes scheduler with the Scheduler Framework plugins, and a 'CAUTION' box warning that Admiralty is not compatible with third-party custom schedulers.

Admiralty Documentation Blog Pricing Contact

GitHub Chat Twitter Search

Introduction
Quick Start
Concepts
Multi-Cluster Scheduling
Operator Guide
User Guide

Multi-Cluster Scheduling

Source Pods

Admiralty's multi-cluster scheduling feature must be enabled at the namespace level with the `multicluster-scheduler=enabled` label, and at the pod level with the `multicluster.admiralty.io/elect=""` annotation. Annotated pods in labeled namespaces are called source pods. Typically, cluster administrators label namespaces, while users annotate pod templates of, e.g., Jobs or Deployments.

Architecture

Multi-cluster scheduling is supported by four components run by pods in Admiralty's installation namespace (typically, `admiralty`):

1. a mutating pod admission webhook,
2. the proxy scheduler,
3. the pod chaperon controller, and
4. the candidate scheduler.

For a given source pod, the mutating pod admission webhook and proxy scheduler are acting in the cluster where the source pod is created, whereas the pod chaperon controller and candidate scheduler are acting in target clusters of that source cluster.

INFO

The proxy and candidate schedulers extend the standard Kubernetes scheduler with [Scheduler Framework](#) plugins. Therefore, they implement all standard Kubernetes scheduling features.

CAUTION

Admiralty is not compatible with third-party custom schedulers. That being said, if your custom scheduler

<https://admiralty.io>

Admiralty's Multicluster-Scheduler

The screenshot shows the GitHub repository page for `admiraltyio / multicluster-scheduler`. The browser address bar shows the URL `https://github.com/admiraltyio/multicluster-scheduler`. The repository has 15 pulls, 215 stars, and 19 forks. The navigation bar includes links for Code, Issues (7), Pull requests (1), Actions, Projects (2), and Security. The main content area shows the `master` branch selected, with buttons for 'Go to file', 'Add file', and 'Code'. A recent commit by `adrienjt` is highlighted, titled 'v0.10.0 changelog and (now scripted) version b...', dated 17 days ago, with 131 comments. Below the commit, a list of files and folders is shown, including `.github/workflows`, `build`, `charts/multicluster-s...`, `cmd`, and `docs`, each with a description and a date. On the right, the 'About' section describes the project as a system of Kubernetes controllers that intelligently schedules workloads across clusters, with links to `admiralty.io`, the README, and the Apache-2.0 License.

Search or jump to... / Pulls Issues Marketplace Explore

admiraltyio / multicluster-scheduler

Unwatch 15 Star 215 Fork 19

<> Code Issues 7 Pull requests 1 Actions Projects 2 Security

master

Go to file Add file Code

adrienjt v0.10.0 changelog and (now scripted) version b... 17 days ago 131

.github/workflows	fix github actions workflow	2 months ago
build	target controller	2 months ago
charts/multicluster-s...	v0.10.0 changelog and (now scripted) versio...	17 days ago
cmd	fix name collisions and truncate names	20 days ago
docs	v0.4.0 see CHANGELOG.md	9 months ago

About

A system of Kubernetes controllers that intelligently schedules workloads across clusters.

admiralty.io

Readme

Apache-2.0 License

Admiralty on Nautilus

- Currently running 0.10.0-rc1
- Have been federating with
 - ARM-based k3s
 - PacificWave Kubernetes cluster
 - Google Cloud Kubernetes cluster
 - Kubernetes Cluster inside Azure
- Getting ready to federate with
 - Expanse's Kubernetes partition
 - A Windows-based Kubernetes cluster

Installing Admiralty

- Pretty well documented in github:
<https://github.com/admiraltyio/multicluster-scheduler/tree/v0.10.0-rc.1>
- Source and target cluster both need Admiralty installed

```
helm install cert-manager ...  
helm install multicluster-scheduler admiralty/multicluster-scheduler ...
```
- Create secret in target cluster and propagate to source cluster

```
(target) kubemcsa export -n klum c1 --as c2 >s.yaml  
(source) kubectl -n admiralty apply -f s.yaml
```
- Whitelist target cluster in source cluster (*helm update ...*)
- You are pretty much good to go!
 - Pods in source cluster just need to add an annotation

```
metadata:  
  annotations:  
    multicluster.admiralty.io/elect: ""
```




Installing Admiralty

- Admiralty creates a set of new resource types

```
Igors-MacBook-Pro:~ isfiligoi$ kubectl api-resources |grep admiralty
```

clustersources	csrc	multicluster.admiralty.io	false	ClusterSource
clustersummaries	mcsun	multicluster.admiralty.io	false	ClusterSummary
clustertargets	ctg	multicluster.admiralty.io	false	ClusterTarget
podchaperons	chap	multicluster.admiralty.io	true	PodChaperon
sources	src	multicluster.admiralty.io	true	Source
targets	tg	multicluster.admiralty.io	true	Target

- Target clusters can be seen as virtual nodes

```
Igors-MacBook-Pro:~ isfiligoi$ kubectl get nodes |grep admiralty
```

admiralty-igor-gke-us-central	Ready	cluster	7d18h
admiralty-k3s	Ready	cluster	7d18h
admiralty-nautilus	Ready	cluster	7d18h

Installing Admiralty

- We have been mostly using one-way federation
 - Nautilus as source, others as targets
- Nautilus can easily be the target, too
 - Admiralty allows for arbitrary mesh
 - Federation with SDSC Expanse is expected to be both ways



Scheduling to target clusters

- Admiralty's Multicloud-Scheduler is a real Kubernetes scheduler
 - Users do not get to pick explicitly the target
 - Offload happens based on standard requirements and preferences
 - Users just have to opt-in
- When there are nodes in multiple possible clusters that match
 - Admiralty will consider only clusters that have free matching nodes
 - Which target cluster will be picked is (mostly) non-deterministic
 - If no target clusters have any available matching nodes, the pod remains pending in the source cluster (only)
- Priorities and preemption work as you would expect them to

Scheduling to target clusters

Under the hood,
uses the standard
k8s filtering and
scoring
mechanisms

Node selection in kube-scheduler

kube-scheduler selects a node for the pod in a 2-step operation:

1. Filtering
2. Scoring

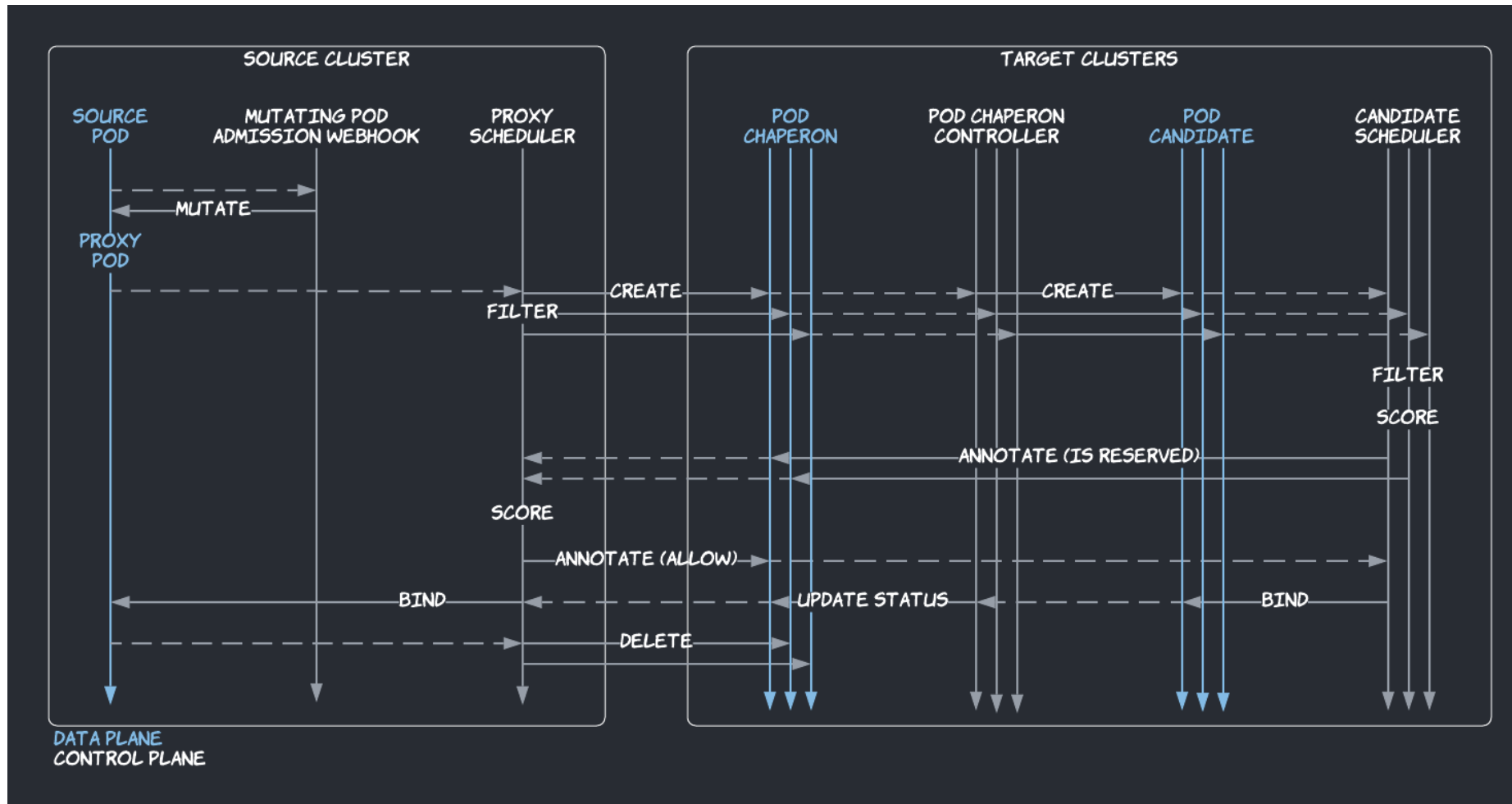
The *filtering* step finds the set of Nodes where it's feasible to schedule the Pod. For example, the PodFitsResources filter checks whether a candidate Node has enough available resource to meet a Pod's specific resource requests. After this step, the node list contains any suitable Nodes; often, there will be more than one. If the list is empty, that Pod isn't (yet) schedulable.

In the *scoring* step, the scheduler ranks the remaining nodes to choose the most suitable Pod placement. The scheduler assigns a score to each Node that survived filtering, basing this score on the active scoring rules.

Finally, kube-scheduler assigns the Pod to the Node with the highest ranking. If there is more than one node with equal scores, kube-scheduler selects one of these at random.

<https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/#kube-scheduler-implementation>

Scheduling to target clusters



Other features

- Admiralty has several other features we have not explored yet
- Three potentially interesting options:
 - Multi-cluster services, using a load-balancing across a Cilium cluster mesh
 - Identity federation (instead of shared secrets)
 - Federation with Targets lacking a public IP (reversed connectivity)

Conclusion

- Admiralty has been in use in the PRP k8s cluster/Nautilus for some time now
- Works as advertised for our main use cases
- We are planning to use it to expand to more clusters in the future

Acknowledgments

- This work was partially funded by the US National Science Foundation (NSF) under grants OAC-1826967, OAC-1541349, MPS-1148698 and OAC-1841530.